Fox Trails Revisited

A hands-on look at developing an eCommerce application using Fox Trails

Requirements Definitions

As with any application development, the most crucial aspect of developing a Fox Trails web site is deciding what you want it to do. For this application we'll borrow some agile development techniques and design it in an iterative fashion. Our initial requirement definition will consist of an outline of the basic functionality that we want the application to have. We'll fill in the details as we go.

First of all, we'll need some products for our web application
Most eCommerce sites provide a list of products to be purchased. Various means are used to navigate to the desired products through some sort of catalog hierarchy.

The ability to select products for purchase
Once our visitors have found the products that they like, they need to be able to select which ones they want to purchase. We'll use a traditional shopping cart for our application.

Accepting Payment
Numerous means are available for accepting payment from a web site. The exact details are usually dependent upon the merchant bank account that you use. For our purposes, we'll simply store the user's credit card number with their order and rely upon a fullfillment application to handle charging the card when the product is shipped. A true eCommerce application would place a hold on the card for the amount of purchase and then convert the hold into a sale at the time of shipment.

Order Fullfillment
A real eCommerce application would also need a way to update online orders with tracking numbers and send emails to customers with all the necessary shipping information. As with credit card processing, fulfillment processing is dependent on the carriers that you use, your accounting & inventory systems and the type of email server that you use. Suffice it to say that this will be an exercise left for you to implement if desired.

Creating a products table

About Migrations
Every Foxpro developer at some point will encounter the dilemma of updating the structure of a database table that has been deployed to a production server. Among the issues that need to be addressed are determining what changes need to be made, handling contention during the update and preserving existing data.

Fox Trails borrows the concept of database migrations from Ruby on Rails. The concept is simple, a migration is a series of instructions that tell Foxpro how to update or revert a database or table. Fox Trails remembers which migrations have been applied, thereby allowing you to apply only the necessary changes to reach a specific version of the database schema.

FoxPro DDL statements
Each migration contains two sets of DDL statements. When a migration is applied, the database will either be upgraded or downgraded to the desired version. Within a migration file, you will see an UP method and a DOWN method. You will typically put create and alter statements into your UP methods and DROP statements into your DOWN methods.

Our first migration

For our first migration, let's create a products table and a category table.

MODIFY COMMAND migrate\002products.prg

```
DEFINE CLASS migrate AS Session
	targetDir = ""
	PROCEDURE Init
		SET CENTURY ON
		SET STRICTDATE TO 0
	ENDPROC

	PROCEDURE Up
		IF FILE(this.targetDir+"products.dbf")
			RETURN
		ENDIF
		CREATE TABLE (this.targetDir+"products.dbf") (;
			productid C(5), title C(40), abstract m,;
			price n(8,2))

		INDEX ON productid TAG productid
		INDEX ON title TAG title
```

```
        USE IN products
    ENDPROC

    PROCEDURE Down
        DROP TABLE (this.targetDir+"products.dbf")
    ENDPROC
ENDDEFINE
```

When finished, apply the migration by running the utility\migrate.prg program and verify that the products table was created.

On your own: Create a migration for the categories table.

You'll want a categoryid and a category title. Apply the migration and verify that your table was created successfully.

Note: applying migrations to production servers.

When applying migrations to production servers, it's always a good idea to make a backup first. You may want to stop your web server if you need to alter the structure of existing tables. This ensures that you will be able to open your tables exclusively. If you are using a SQL database, this may not apply to you.

Entering Product Data

Scaffolding
Fox Trails includes a utility to create a simple maintenance set for entering data into a table. DO utility\scaffold to create the necessary files.

Open your web browser to http:\\localhost\default.aspx\product and enter a few products. You may need to ensure that you have write permissions to the DB subdirectory. Ensure that the user specified in IIS's Directory Security has write access to the directory. Also ensure that ASP.Net's Application tab is set to Local Impersonation.

Go ahead and add a few category records as well.

NOTE: Handling existing data
When developing a Fox Trails application, you may have existing data that you would like to use. If your Fox Trails application is hosted at your client's site, you may be able to access the existing data directly. Another option is to use CursorToXml for performing bulk data loads of a remote Fox Trails server.

Here's some sample code that will upload a xml file to a foxtrails server. It's untested and is provided here for reference only.

```
CURSORTOXML("inventory","lcXml",1,1,0,"")
oXHR = new CREATEOBJECT("MSXML2.XMLHTTP");
oXHR.open("POST", "http://foxtrailserver/default.aspx/product/load", false);
oXHR.send("xmldata="+URLENCODE(lcXml));
```

This code will send an xml file to the Fox Trails server's product controller and call it's load method. The xml data will be attached to the product controller's xmldata property.

In the load method of the product controller, do an XmlToCursor and then append the data to your destination table.

Displaying Our Products

Creating an online catalog

Now that we have some products and categories, we need to create a way to display them to our visitors. We'll need a catalog controller and layout. Although we could use a scaffold to generate these files, the desired functionality doesn't fit a crud maintenance set very well. It would take more work to retrofit the scaffold than creating them manually.

When a user clicks on our catalog, we'll simply display a list of products. By convention, the index method is called on a controller if the method name is not passed in the url. If we have time, we'll look at modifying our catalog to use the categories table that we created.

Go ahead and create the following controller.

Controllers\catalog.prg

```
DEFINE CLASS catalog AS base OF Controllers\base.prg
	PROCEDURE opentables
		USE DB\products ORDER TAG title
	ENDPROC

	PROCEDURE index
		*-- we don't need to do anything here, yet.
	ENDPROC
ENDDEFINE
```

Now that we have the controller, we'll need to create a template to display it. When creating templates, don't get too elaborate. You can always make it look better later. The goal right now is just to get the core functionality working.

If you've never created a web page before, it may help to think of an html file as simply a text file with formatting codes, called "html tags". Most html tags are paired, consisting of a start tag, then the text to be formatted, followed by the closing tag. Here is an example **This text is bold**. If you've ever used WordPerfect's reveal codes feature, html is structured in a similar way.

Typing html by hand can be a bit intimidating. Thankfully, you don't have to. Because FoxTrails separates html from your application logic, you can use any html editor to create your vml files. The industry standard for editing Html files is Dreamweaver. You can also use Visual Studio to edit vml files.

Layouts\catalog.vml

```
<html>
<head><title>ACME Hardware</title></head>
<body>
<%
SELECT products
SCAN
%>
<table width="100%">
<tr><th><<products->title>></th>
      <th width="100" align="right"><<transform(products->price,"$999.99")>></th>
</tr>
<tr><td colspan="2"><<products->abstract>></td></tr>
</table>
<hr/>
<% ENDSCAN %>
</body>
</html>
```

When you are finished, open http://localhost/default.aspx/catalog in your web browser and verify that your products show up. If you see a blank page, take a look at the foxtrails.err file.

NOTE:
Wysiwyg Html Editors may complain about Foxpro's standard textmerge delimiters. These can be changed to anything you want in the base controller's init method. If you decide to change the delimiters, you might want to use a delimiter that most editors understand such as php's <? ?>.

The Shopping Cart

Now that we have a list of products, we need a create a table to store our customer's cart in. What exactly should we put in the cart? We can store just the product id and quantity, or we can add the price as well. It's really a matter of personal preference. Although it may be easier to store the price in the cart, doing so will complicate additional features such as allowing users to save their cart or email it to friends. If you have a price change, you would need to update the price in any saved carts. Let's avoid that issue for now and lookup the current price when we need to display it.

Migrate\004carts.prg

```
DEFINE CLASS migrate AS Session
	targetDir = ""
	PROCEDURE Init
		SET CENTURY ON
		SET STRICTDATE TO 0
	ENDPROC

	PROCEDURE Up
		IF FILE(this.targetDir+"carts.dbf")
			RETURN
		ENDIF
		CREATE TABLE (this.targetDir+"carts.dbf") (;
			sid C(36), seq c(3), productid C(5), qty n(3))

		INDEX ON sid TAG sid
		INDEX ON sid+seq TAG sortkey
		INDEX ON sid+productid TAG productkey

		USE IN carts
	ENDPROC

	PROCEDURE Down
		DROP TABLE (this.targetDir+"products.dbf")
	ENDPROC
ENDDEFINE
```

Apply the migration and verify that the carts.dbf table was created successfully. Although we are using free tables for this demonstration, you could just as easily create a database container and use local or remote views, sql passthrough or cursor adapters.

Adding products
Now that we have a shopping cart to store items in, we need to be able to add something to it. Let's write some code to add products to our cart.

Controllers\cart.prg

```
DEFINE CLASS cart AS base OF Controllers\base.prg
	PROCEDURE opentables
		USE DB\products ORDER TAG productid IN 0
		USE DB\carts IN 0
		SELECT carts
	ENDPROC

	PROCEDURE addProduct
		LPARAMETERS lcProductId

		SELECT carts
		SET ORDER TO TAG productkey
		IF SEEK(this.sid + lcProductId)
			REPLACE qty WITH qty + 1
		ELSE
			SET ORDER TO TAG sortkey DESCENDING
			IF SEEK(this.sid)
				lcSeq = TRANSFORM(VAL(carts->seq)+1,"@L 999")
			ELSE
				lcSeq = "001"
			ENDIF

			INSERT INTO carts ;
				(sid,seq,productid,qty) ;
			VALUES ;
				(this.sid,lcSeq,lcProductId,1.00)
		ENDIF
		*-- display our shopping cart
		this.index()
	ENDPROC

	PROCEDURE index
		SELECT carts
		SET ORDER TO TAG sortkey
		SET RELATION TO productid INTO products
		=SEEK(this.sid)
		oDispatchContext.RenderAction = "index"
	ENDPROC
ENDDEFINE
```

Next we need to modify the catalog display with an add to cart link. Add the highlighted lines to the catalog.vml template. You'll notice that we're using a few utility methods to make our job a little easier. The linkto function creates an html hyperlink. We pass it the url for our addProduct method and the name to put on the link. The urlencode is needed any time you pass user defined values in the url.

Layouts\catalog.vml

```
<html>
<head><title>ACME Hardware</title></head>
<body>
<%
SELECT products
SCAN
        lcLink = Linkto( "/cart/addProduct/"+urlencode(products->productid), ;
                        "Add to Cart")
%>
<table width="100%">
<tr><th><<products->title>></th>
      <th width="100" align="right"><<transform(products->price,"$999.99")>></th>
</tr>
<tr><td colspan="2"><<products->abstract>></td></tr>
<tr><td><<lcLink>></td></tr>
</table>
<hr/>
<% ENDSCAN %>
</body>
</html>
```

At this point, we should be able to add products to our shopping cart. Open http://localhost/default.aspx/catalog and click on one of the links. Although we have not added any templates to display our cart, you can open the carts table in Foxpro and verify that it is working as desired.

Displaying our cart

Next we need to create a template to display our shopping cart. We'd also like to be able to edit the quantities on our shopping cart. We'll use some ajax to update our shopping cart without forcing a reload of the cart page. Because we will be updating portions of the cart page with ajax, we will use both layouts and views to generate the html for our shopping cart. Fox Trails makes it easy to embed a view into it's layout. Simply add <<oDispatchContext.RenderedView>> to the layout where you want the view displayed.

Layouts\cart.vml

```
<html>
<head><title>ACME Hardware</title></head>
<body>
<<oDispatchContext.RenderedView>>
</body>
</html>
```

Now we'll create the template to display the shopping cart view. You'll notice a bit more logic here. We need to create unique id's for each field that we want to access using Ajax. We'll append the cart's sequence number to each element to make it unique.

We will also do a few calculations in Foxpro, although we could also total these using Ajax. We'll save that for later, when we implement our Ajax logic.

Views\cart\index.vml

```
<table>
<tr><th>Product</th><th>Quantity</th><th>Price</th></tr>
<% SELECT carts
   lnTotal = 0
   SCAN FOR sid = cart.sid
        lcRowId = "row"+carts->seq
        lcQtyId = "qty"+carts->seq
        lcPriceId = "price"+carts->seq
        lcPriceDisplay = "price_cell"+carts->seq
        lnExtendedPrice = products->price * carts->qty
        lnTotal = lnTotal + lnExtendedPrice
%>
<tr id="<<lcRowId>>" <<iif(carts->qty = 0,[style="display:none;"],"")>> >
   <td><<products->title>></td>
   <td>
        <input id="<<lcQtyId>>" name="_n<<lcQtyId>>" maxlength="3"
                size="4" value="<<carts->qty>>" />
        <input id="<<lcPriceId>>" type="hidden" value="<<products->price>>" />
   </td>
   <td id="<<lcPriceDisplay>>"><<lnExtendedPrice>></td>
</tr>
<% ENDSCAN %>
<tr><td> </td><th>Total Price</th><th id="total"><<lnTotal>></th></tr>
</table>
```

Adding Ajax
When working with Ajax, it's helps to take small steps, changing too many things at once will make it difficult to troubleshoot if something doesn't work the way you expect.

First of all, let's start with adding the ajax library and use Ajax to write some text into the page.

Layouts\cart.vml
```
<html>
<head><title>ACME Hardware</title></head>
<script type="text/javascript" src="/lib/prototype.js"></script>
<script type="text/javascript" src="/lib/cart.js"></script>
<body>
<<oDispatchContext.RenderedView>>
<span id="sayhello">Ajax is not working</span>
</body>
</html>
```

WebRoot\lib\cart.js
```
Event.observe(window,"load", function() {
        $('sayhello').update("Hello from Prototype");
});
```

Open the cart http://localhost/default.aspx/cart and verify that Prototype was loaded successfully. If you don't see "Hello from Prototype", ensure that you've used the right paths and that you've typed everything correctly. If you plan on doing extensive work with Ajax, I would recommend installing the Fire Bug plugin in Fire Fox.

Now that we have Prototype installed, let's add the rest of our Ajax logic. We need to add another field to the index template that stores how many products are in the cart. Make the highlighted modifications to index.vml. Go ahead and remove the span from the cart.vml layout as well.

Views\cart\index.vml
```
   lnTotal = 0
   lnLastSeq = 0
   SCAN FOR sid = cart.sid
        ...
        lnTotal = lnTotal + lnExtendedPrice
        lnLastSeq = VAL(carts->seq)
%>
...
</table>
<input id="lastSeq" type="hidden" value="<<lnLastSeq>>" />
```

Next we need to modify cart.js to update the extended price of each product when it's quantity changes. For now, trust me on these changes. There are numerous web sites and books on Prototype's Ajax library, if you'd like to understand how all of this works.

WebRoot\lib\cart.js

```
Event.observe(window,'load', function() {
	var productCount = parseInt($('lastSeq').value);

	// hook up our onchange event
	$R(1,productCount).each( function( seq ) {
		var seqStr = '00'+seq;
		seqStr = seqStr.substr(seqStr.length-3);
		var qtyElement = $('qty'+seqStr);
		qtyElement.onchange = updatePrice.bind(qtyElement,seqStr);
	});

	// this will allow us to reload the page
	// if our ajax callback fails for some reason
	Ajax.Responders.register({
		onFailure: function() {
			document.location = "/default.aspx/cart";
		}
	});

});

function updatePrice(seqStr) {
	// calculate the new extended price
	var qty = parseInt($F(this));
	var price = parseFloat($F('price'+seqStr));
	var extendedPrice = qty * price;

	// update the selected row
	var priceDisplay = $('price_cell'+seqStr);
	var originalPrice = parseFloat(priceDisplay.innerHTML);
	if ( qty == 0 ) {
		$('row'+seqStr).hide();
	} else {
		priceDisplay.update(''+extendedPrice);
	}

	// update our cart total
	var diff = extendedPrice - originalPrice;
	var total = parseFloat($('total').innerHTML) + diff;
	$('total').update(''+total);
```

```
        // now that we've updated our display, let's inform our
        // controller of the change with a callback function
        Ajax.Request("/default.aspx/cart/update/"+seqStr+"/"+qty);

        // if we've removed everything from the cart
        // reload the page
        if ( total == 0 ) {
                document.location = "/default.aspx/cart";
        }

}
```

The Ajax.Request line above sends a callback to the cart controller with the sequence number that was updated and the new quantity. We need to add an update method to the controller. While we're at it, let's handle an empty cart as well.

Controllers\cart.prg

```
DEFINE CLASS cart AS base OF Controllers\base.prg
        PROCEDURE opentables
                USE DB\products ORDER TAG productid IN 0
                USE DB\carts IN 0
                SELECT carts
        ENDPROC

        PROCEDURE addProduct
                LPARAMETERS lcProductId

                SELECT carts
                SET ORDER TO TAG productkey
                IF SEEK(this.sid + lcProductId)
                        REPLACE qty WITH qty + 1
                ELSE
                        SET ORDER TO TAG sortkey DESCENDING
                        IF SEEK(this.sid)
                                lcSeq = TRANSFORM(VAL(carts->seq)+1,"@L 999")
                        ELSE
                                lcSeq = "001"
                        ENDIF

                        INSERT INTO carts ;
                                (sid,seq,productid,qty) ;
                        VALUES ;
                                (this.sid,lcSeq,lcProductId,1.00)
                ENDIF
                *-- display our shopping cart
```

```
        this.index()
    ENDPROC

    PROCEDURE index
        SELECT carts
        SET ORDER TO TAG sortkey
        SET RELATION TO productid INTO products
        *-- check if we have an empty cart
        SUM qty TO lnTotalQty FOR sid = this.sid
        SEEK this.sid
        IF lnTotalQty = 0
            oDispatchContext.RenderAction = "empty"
        ELSE
            oDispatchContext.RenderAction = "index"
        ENDIF

    ENDPROC

    PROCEDURE update
        LPARAMETERS lcSeq, lcQty

        SELECT carts
        IF SEEK(this.sid+lcSeq,"carts","sortkey")
            REPLACE qty WITH VAL(lcQty)
        ELSE
            *-- return an error code here
        ENDIF
        oDispatchContext.RenderLayout = ""
    ENDPROC
ENDDEFINE
```

We'll also add a special view to display an empty cart.

Views\Cart\empty.vml

```
<input id="lastSeq" type="hidden" value="0" />
<strong>Your shopping cart is empty</strong>
```

Accepting Payment

Our last task is to create the ability to accept payments from our customers. We'll need a migration, a controller and a layout.

Migrate\005orders.prg

```
DEFINE CLASS migrate AS Session
	targetDir = ""
	PROCEDURE Init
		SET CENTURY ON
		SET STRICTDATE TO 0
	ENDPROC

	PROCEDURE Up
		IF FILE(this.targetDir+"orders.dbf")
			RETURN
		ENDIF
		CREATE TABLE (this.targetDir+"orders.dbf") (;
			sid C(36), total n(8,2), ccname c(30), ccnumber c(17), ;
			ccexpmon c(3), ccexpyr c(4), address c(30), ;
			city c(25), state c(2), zip c(5), phone c(15) )

		INDEX ON sid TAG sid

		USE IN orders
	ENDPROC

	PROCEDURE Down
		DROP TABLE (this.targetDir+"orders.dbf")
	ENDPROC
ENDDEFINE
```

Layouts\checkout.vml

```
<html>
<head><title>ACME Hardware</title></head>
<body>
<<oDispatchContext.RenderedView>>
</body>
</html>
```

Controllers\checkout.prg

```
DEFINE CLASS checkout AS base OF Controllers\base.prg
	PROCEDURE opentables
		USE DB\products ORDER TAG productid IN 0
		USE DB\carts ORDER TAG sortkey IN 0
		SELECT carts
		SET RELATION TO productid INTO products

		USE DB\orders ORDER TAG sid IN 0
		SELECT orders
		this.update()

	ENDPROC

	PROCEDURE update
		SELECT orders
		IF SEEK(this.sid)
			SCATTER NAME this ADDITIVE
		ELSE
			lcSid = this.sid
			SCATTER NAME this BLANK ADDITIVE
			this.sid = lcSid
		ENDIF
	ENDPROC

	PROCEDURE placeOrder
		SELECT carts
		SUM qty * products->price TO lnTotal FOR sid = this.sid
		this.total = lnTotal

		SELECT orders
		IF SEEK(this.sid)
			GATHER NAME this
		ELSE
			INSERT INTO orders FROM NAME this
		ENDIF
	ENDPROC
ENDDEFINE
```

We also need to add two views, one to accept the payment information and the other to display the customer's receipt.

Views\Checkout\index.vml

```
<form action="/default.aspx/checkout/placeOrder" method="post">
<table>
<<fieldfor("orders->ccname","Card Holder")>>
```

```
<<fieldfor(“orders->ccnumber”,”Card Card #”)>>
<<fieldfor(“orders->ccexpmon”,”Expiration Month”)>>
<<fieldfor(“orders->ccexpyr”,”Expiration Year”)>>
<<fieldfor(“orders->address”,”Billing Address”)>>
<<fieldfor(“orders->city”,”Billing City”)>>
<<fieldfor(“orders->state”,”Billing State”)>>
<<fieldfor(“orders->zip”,”Billing Zip”)>>
<<fieldfor(“orders->phone”,”Phone Number”)>>
<tr><td></td><td><input type=”submit” name=”submit” value=”Place Order” />
</table>
</form>
```

Views\Checkout\placeOrder.vml

```
<h1>Thank you for your order.</h1>
<strong>Your credit card will be charged when your order is shipped.</strong>
<hr/>
<table>
<tr><th>Product</th><th>Quantity</th><th>Price</th></tr>
<% SELECT carts
   lnTotal = 0
   SCAN FOR sid = checkout.sid
        lcRowId = "row"+carts->seq
        lcQtyId = "qty"+carts->seq
        lcPriceId = "price"+carts->seq
        lcPriceDisplay = "price_cell"+carts->seq
        lnExtendedPrice = products->price * carts->qty
        lnTotal = lnTotal + lnExtendedPrice
%>
<tr id="<<lcRowId>>" <<iif(carts->qty = 0,[style=”display:none;”],””)>> >
   <td><<products->title>></td>
   <td><<carts->qty>></td>
   <td><<lnExtendedPrice>></td>
</tr>
<% ENDSCAN %>
<tr><td> </td><th>Total Price</th><th id="total"><<lnTotal>></th></tr>
</table>
<hr/>
<strong>Print this page for your records</strong>
```

Add a link at the bottom of the cart to call our checkout process

Views\Cart\index.vml

```
<<LinkTo(“/checkout”,”Proceed to Checkout”)>>
```

Open your web browser and try out the completed eCommerce site.
http://localhost/default.aspx/catalog

Sprucing Things Up

As you have no doubt noticed, the site that we have just created is a very plain looking site. While we have succeeded in creating a functional eCommerce site, it will not be very successful without a dramatic facelift. Here are a few ideas for sprucing it up.

Include a stylesheet in the layouts
Stylesheets are more powerful than ever and can transform almost any html page into a work of art. As mentioned earlier, vml templates can be edited using tools like DreamWeaver or Visual Studio. There are also various stylesheet tutorials on the web.

Add some images to the site
Add a logo to the site, add another field to the products table to store the url for your product images.

Add Ajax validations to the checkout process
Add tests to ensure that names, cc numbers and addresses are correct. For instance, a Card Holder name should be longer than 2 digits and a credit card number should validate against it's check digit (it's called a luhn algorithm).

Add Formatting for data entry fields
Replace the expiration month and year with combo boxes, format the phone number.

Conclusion

In this course, my primary objective was to help developers get acquainted with the capabilities of Fox Trails by assisting them with the development of an extensive example. It is my hope that you will continue to experiment with Fox Trails and consider how you might use it for your next web development project.